zkLogin: Privacy-preserving blockchain authentication with existing credentials

Foteini Baldimitsi | Kostas Chalkias | Yan Ji | Jonas Lindstrøm | Deepak Maram | Ben Riva | **Mahdi Sedaghat** | Arnab Roy | Joy Wang

PostDoc at Cosic, KU Leuven
Co-Founder at Soundness Labs (soundness.xyz)

ACM CCS'24, SBC'24, RWC'25

17/03/2025, O1Labs Seminars

Mysten Labs and Sui:

Deployed on

zkLogin after ~1 year?

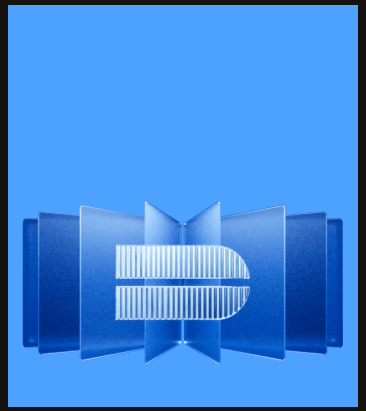
One of the widely used zkApps to date

It has been used for over 7.6 million transactions.
With around 2.4 million unique proofs (March 14).

Sui (L1)

Walrus (DDA)

Move
(Rust Smart Contract)



Deepbook
(DeFi)

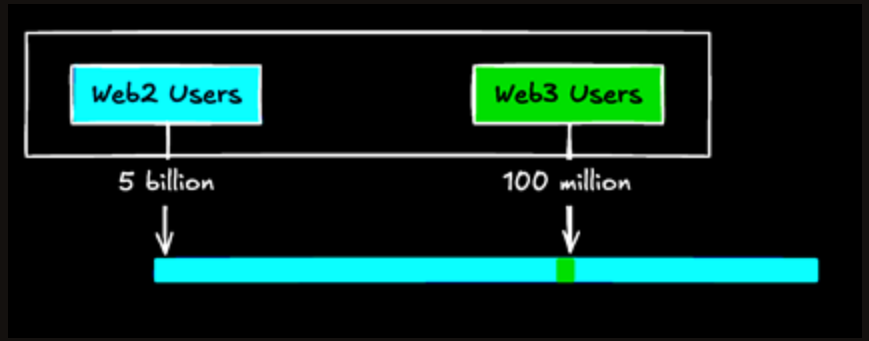
Enoki

There are around

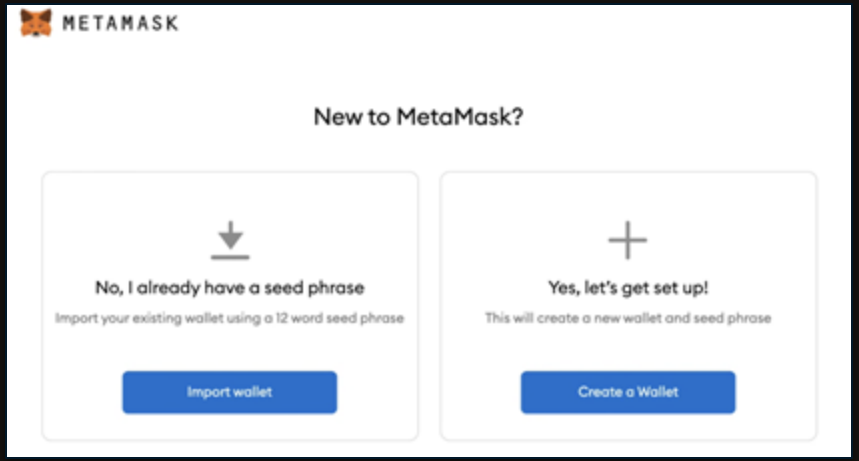
100 million

active crypto wallets

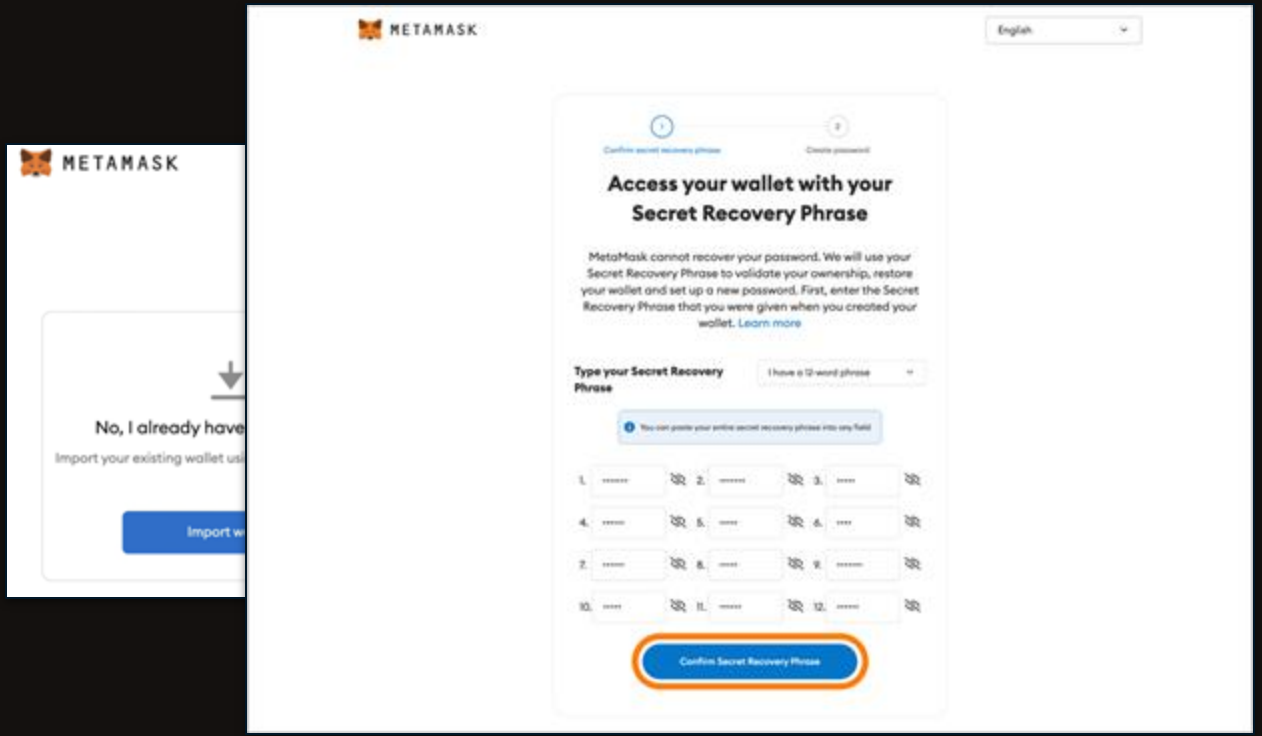
and there are several
BILLIONS
of web2 accounts



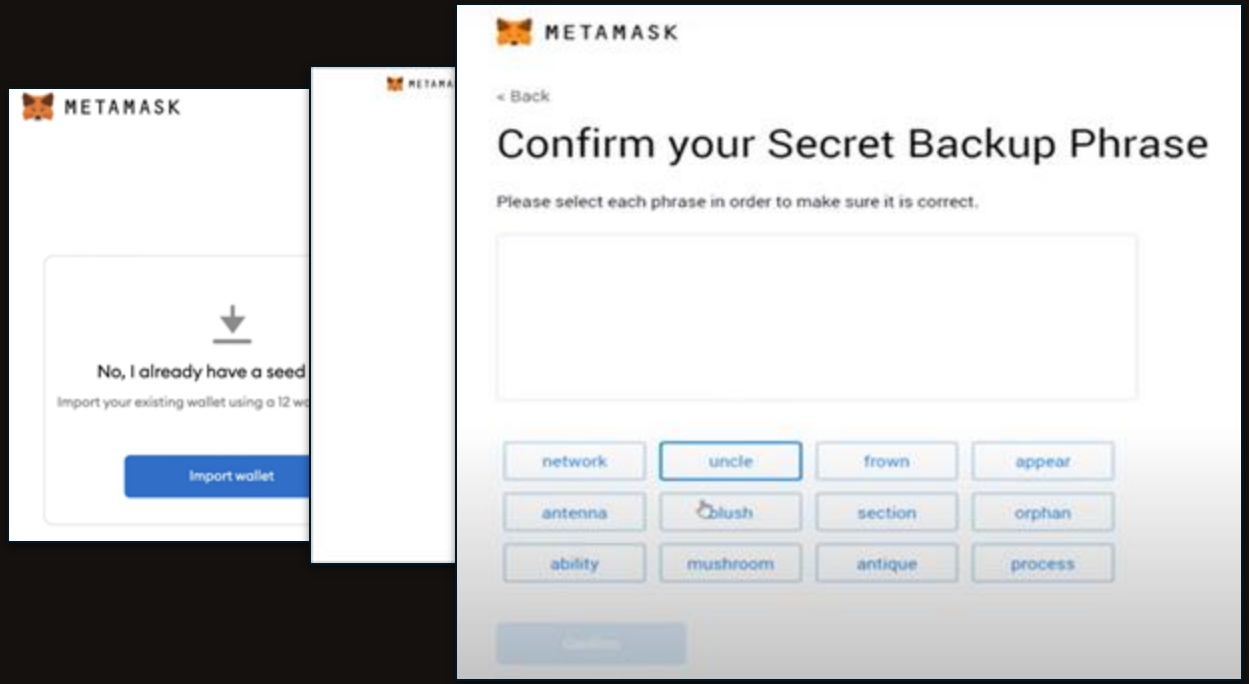
Web3 has an onboarding problem



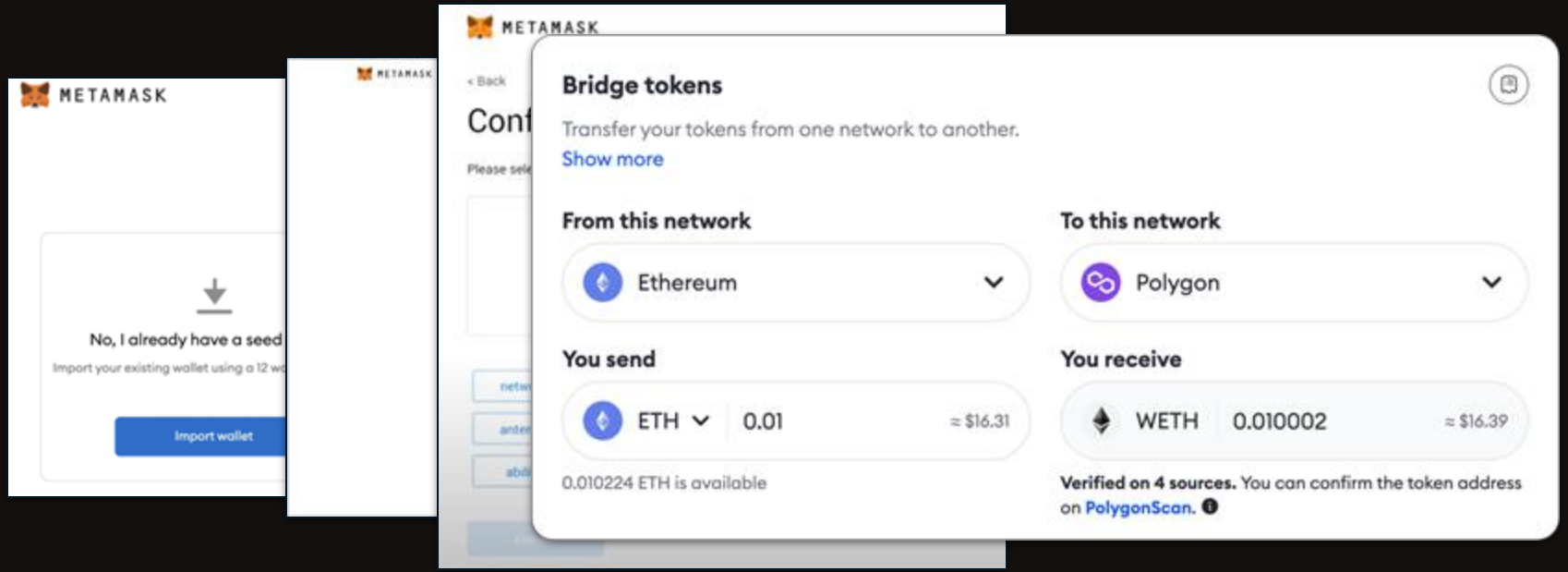
Web3 has an onboarding problem



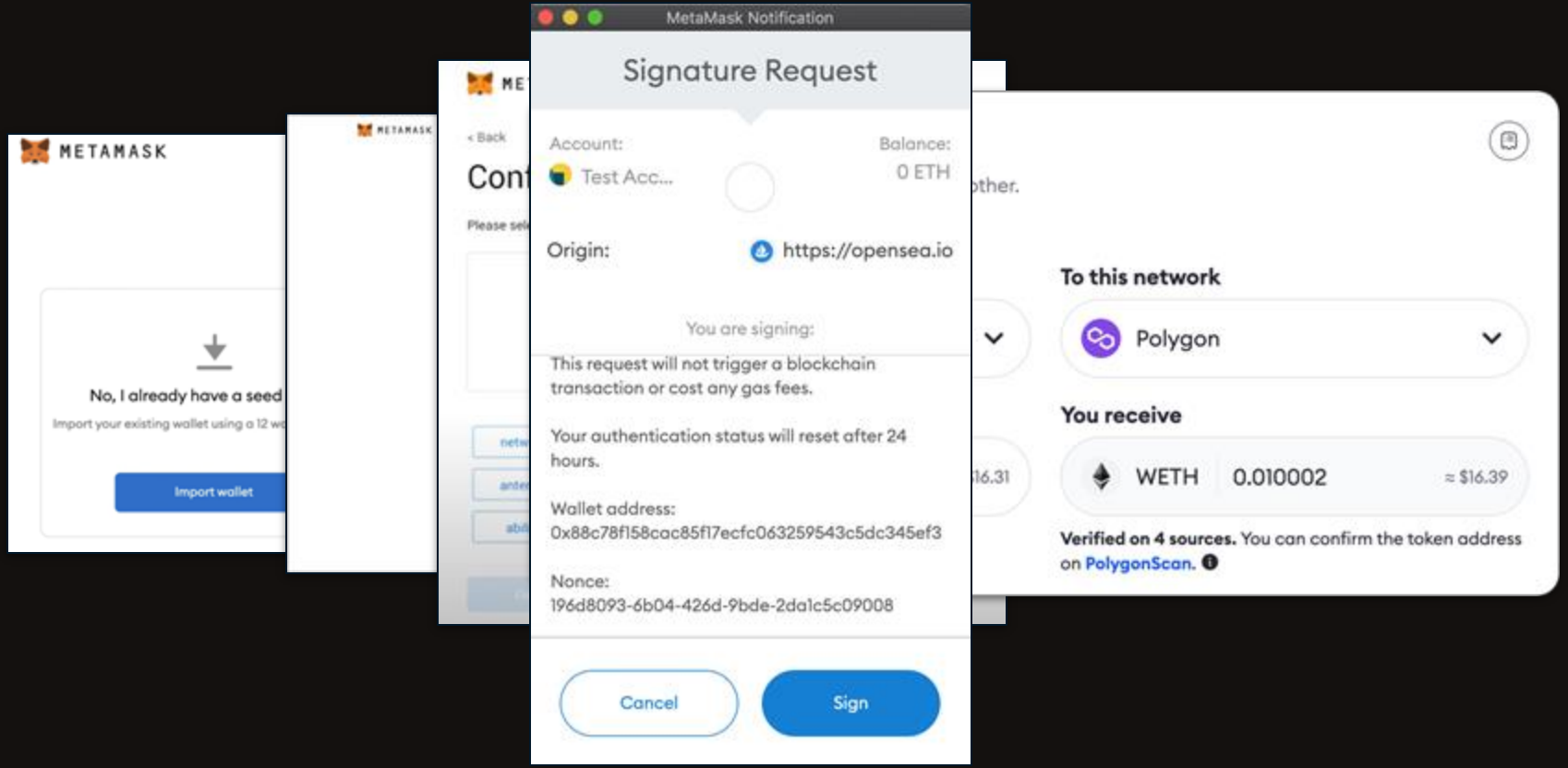
Web3 has an onboarding problem



Web3 has an onboarding problem



Web3 has an onboarding problem

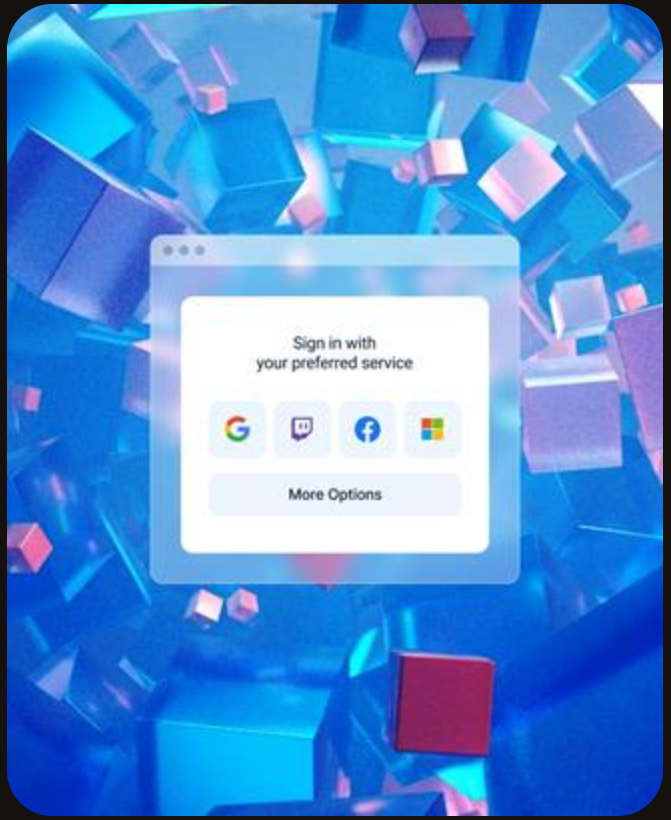


Mnemonics and keys are not going to get us mass adoption.

Complexity is the killer of adoption.
The ultimate killer dApp for blockchain, is accessibility.

Can we make it as easy as signing in with Google, Facebook and co?

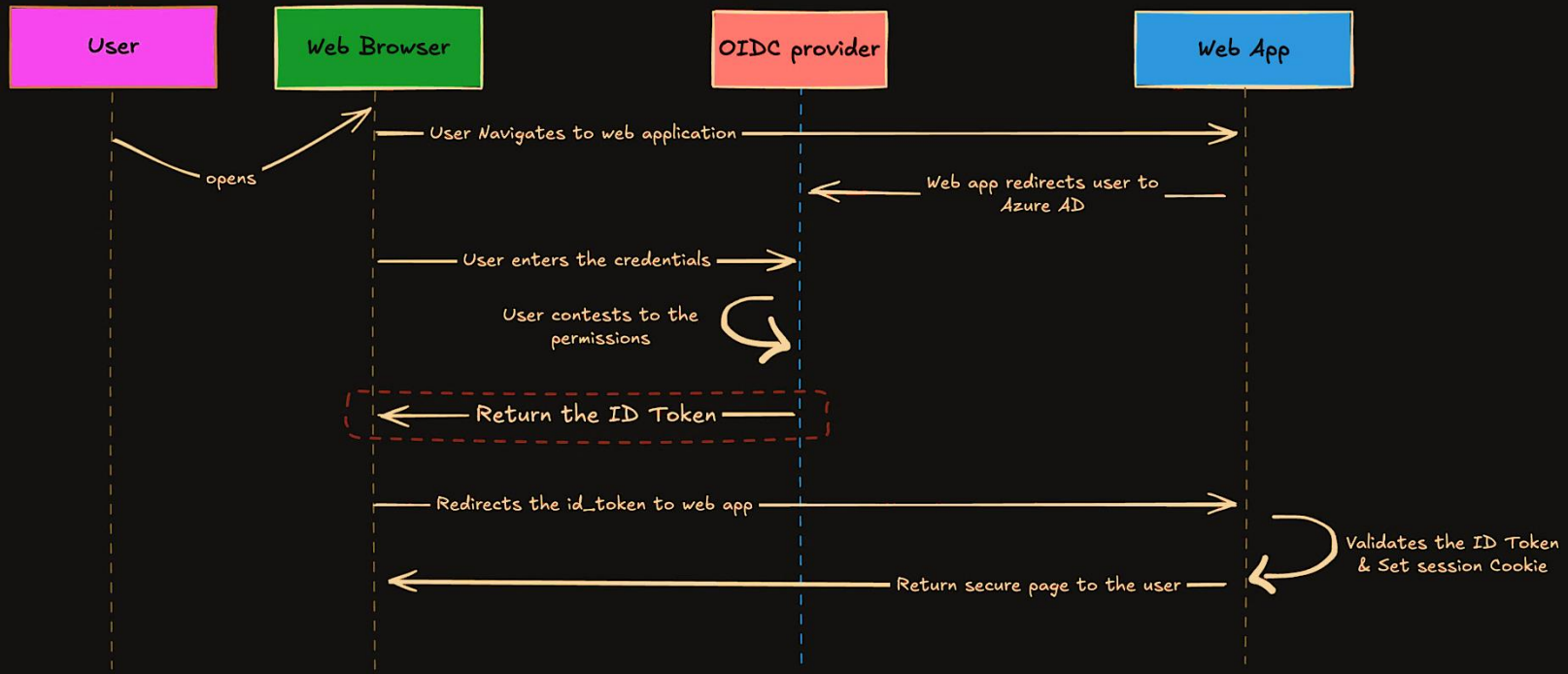
- People don’t want to use separate passwords for each and every app, each and every web2 service
- Extremely likely they already have a Google, Facebook, Amazon account
- Solution: use OAuth to leverage these already existing accounts



zkLogin:
OAuth + Zero Knowledge Proof

Non-custodial
User-friendly
Privacy-preserving

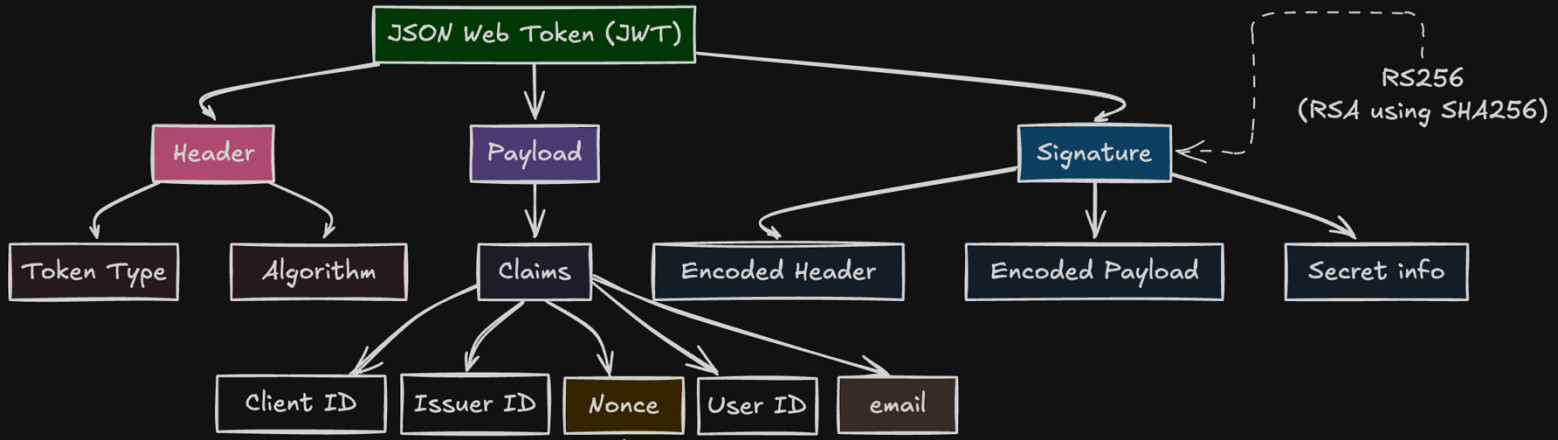
OpenID Connect (an extension of OAuth 2.0)



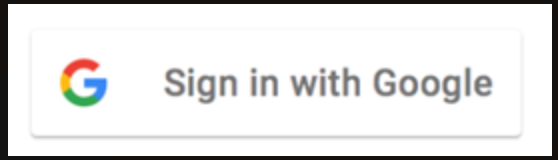
JWT: JSON Web Token

Base64-encoded, RSA-signed

JWT as an alternative to a private key?



A Google-issued JWT (decoded)



Header


```
{
   "alg" : "RS256"
   "kid" : "9697180879682 9a972e79a9d1a9fff11cd61b1e3",
   "typ": "JWT"
}
```


Payload


```
{
   "iss": "https://accounts-google.com",
   "azp": "http://575519204237-msop9ep45u2u098hapqmngv8d84qdc8k-apps.googleusercontent.com",
   "aud": "http://575519204237-msop9ep45u2u098hapqmngv8d84qdc8k-apps.googleusercontent.com",
   "sub": "1104634521",
   "nonce": [redacted],
   "iat": 1682002642,
   "exp": 1682002642,
   "jti": "a8a0728a3ffd5d81ecfd0ea81d0d33d803eb830",
   "email": "test@soundness.xyz"
}
```


you can ask for email
and other personal info

Inject a fresh public key into JWT!

Payload

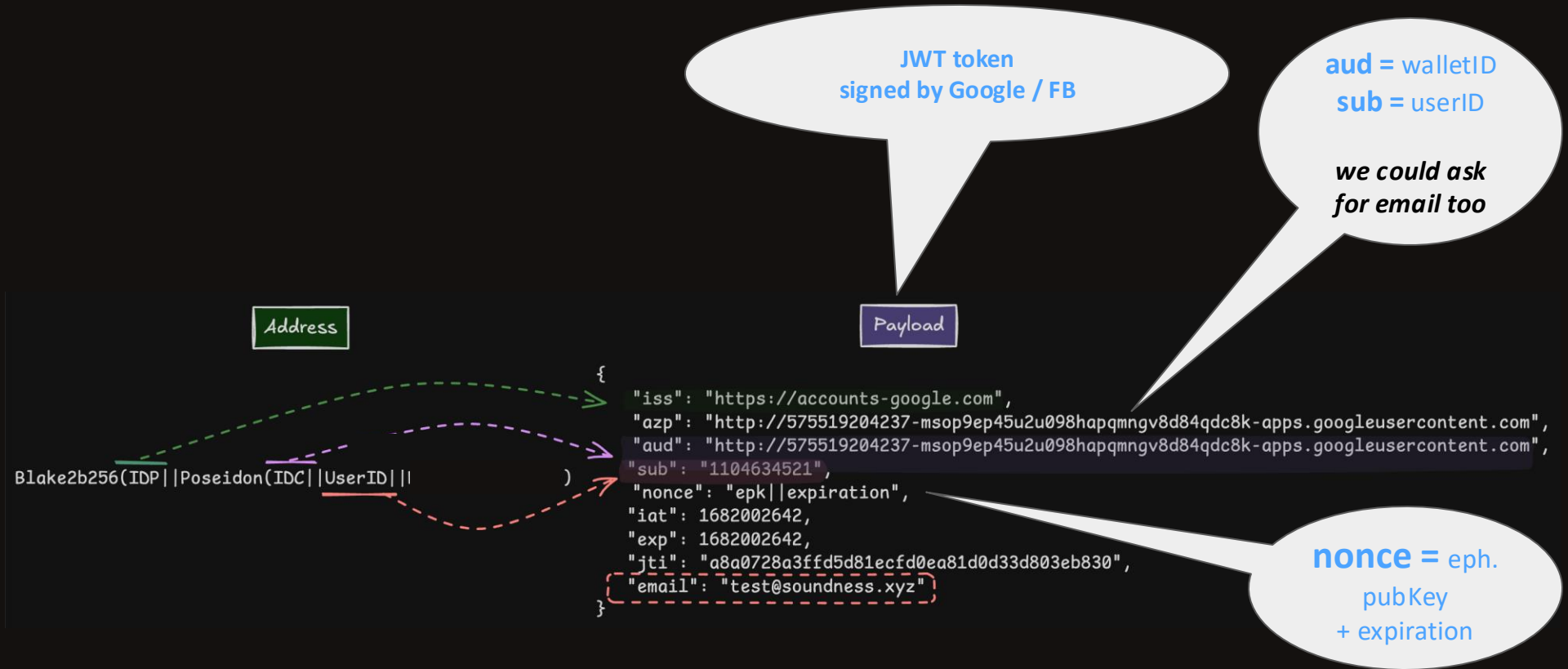

```
{
  "iss": "https://accounts-google.com",
  "azp": "http://575519204237-msop9ep45u2u098hapqmngv8d84qdc8k-apps.googleusercontent.com",
  "aud": "http://575519204237-msop9ep45u2u098hapqmngv8d84qdc8k-apps.googleusercontent.com",
  "sub": "1104634521",
  "nonce": "epk||expiration",
  "iat": 1682002642,
  "exp": 1682002642,
  "jti": "a8a0728a3ffd5d81ecfd0ea81d0d33d803eb830",
  "email": "test@soundness.xyz"
}
```


replace ***nonce*** with user provided data:

ephemeral pub key + expiration

We have a DIGITAL CERT over our fresh key + expiration

zkLogin tricks:



How to ensure users' privacy?

Blake2b256(IDP||Poseidon(IDC||UserID||Poseidon(Salt)))

Address

Add a persistent randomizer: salt

Salt: A persistent per-user secret for **unlinkability**

How to hide the JWT?

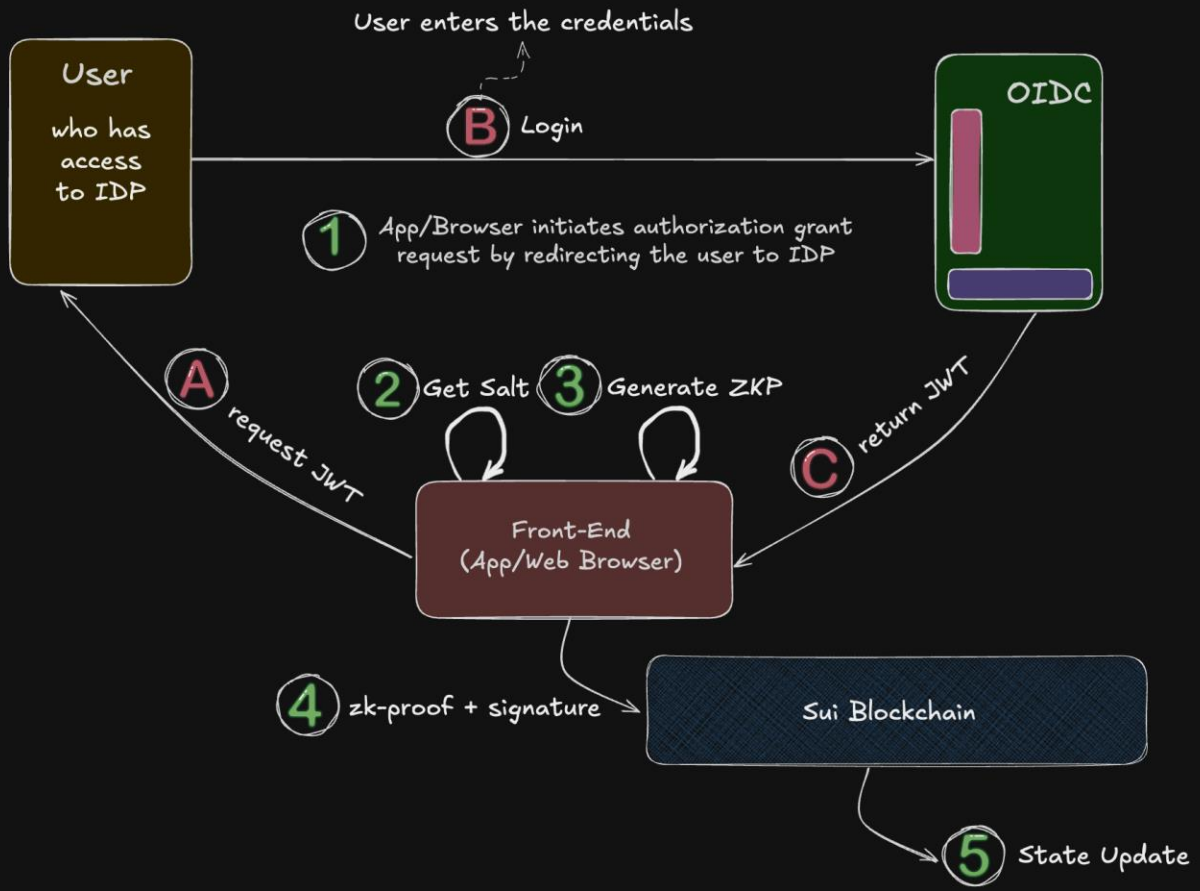
SNARKs to the rescue!

Goal: Prove you have a valid JWT + you know the salt + you injected the ephemeral key into JWT

- Verify JWT's signature using Google's public key
- Verify the ephemeral public key is injected into the JWT's nonce
- Verify that the address is derived correctly from the JWT's userID, walletID, providerID + user's salt

Given a public IDP_pk and zkLogin address:
I have access to a valid JWT under IDP_pk such that:
zkLogin_add = Blake2b256(iss||Poseidon(aud||sub||Poseidon(Salt))) &
Signature on tnx details is valid under epk that is linked to JWT.

zkLogin in one slide: e2e



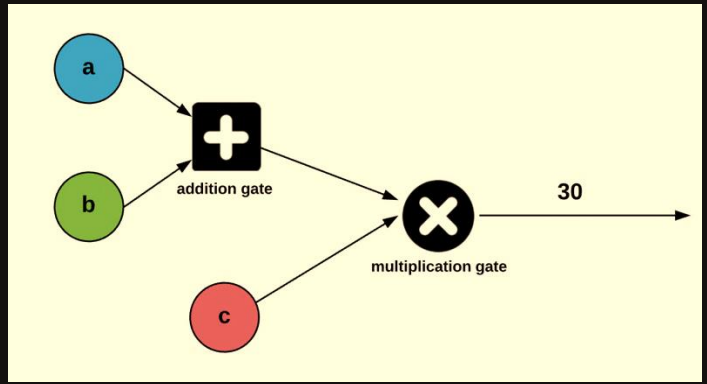
Who maintains the salt?

- Client-side on-device management
  - Edge cases, e.g., cross-device sync, device loss need handling

- Server-side management by a “salt service”
  - Each wallet can maintain their own service/delegate it
  - Privacy models: Store salt either in TEE/MPC/plaintext
  - Auth policies to the service: Either JWT or 2FA

Circuit details

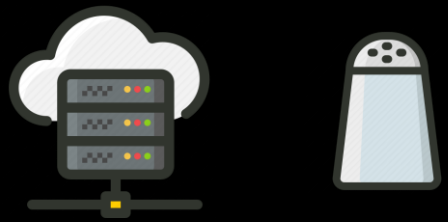
- Implemented in Circom DSL: ~1M R1CS constraints
- We chose Groth16 due to its small proofs + rich ecosystem + fast prover
- Key operations
  - SHA-2 (66%)
  - RSA signature verification (14%) using tricks from [KPS18]
  - JSON parsing, Poseidon hashing, Base64, extra rules (20%)
- Prover based on rapidsnark
  - C++ and Assembly based



zkLogin latency

These numbers correspond only to the **first transaction of a session**

Salt service on AWS Nitro enclave (m5.xlarge10: 4 vCPUs, 16GB RAM)



ZKP generation on Google Cloud (n2d-standard-16: 16 vCPUs, 64GB RAM).

Operation	zkLogin	Ed25519
Fetch salt from salt service	0.2 s	NA
Fetch ZKP from ZK service	2.78 s	NA
Signature verification	2.04 ms	56.3 μs
E2E transaction confirmation	3.52 s	120.74 ms

Latency for most zkLogin transactions is **very similar** to traditional ones!

zkLogin trade-offs

Prover Service

Prover sees JWT; risks unlinkability between web2 and web3 identities.

Time-consuming on most devices, but proofs can be cached.

Local Proof Generation

Is zkLogin really Non-Custodial?

App-Managed Salt

App can break unlinkability, posing potential risks.

Users manage an additional secret, which is less sensitive than a mnemonic.

User-Managed Salt

The option of multi-sig option:

Involve more IDPs instead of one

zkLogin

single-click accounts w/

- Google
- Facebook
- Twitch
- Apple
- Slack
- Microsoft

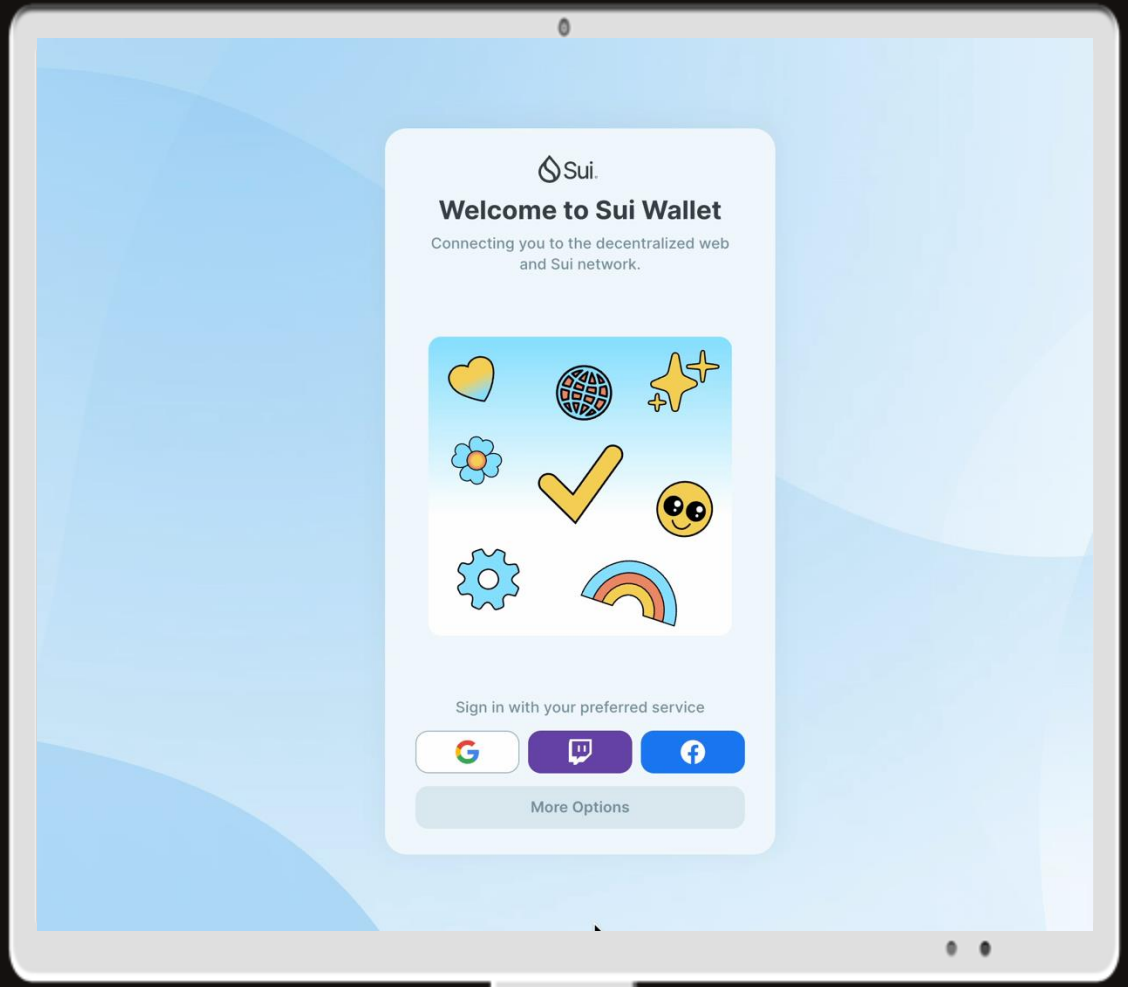
native authenticator

non-custodial

*discoverable, claimable

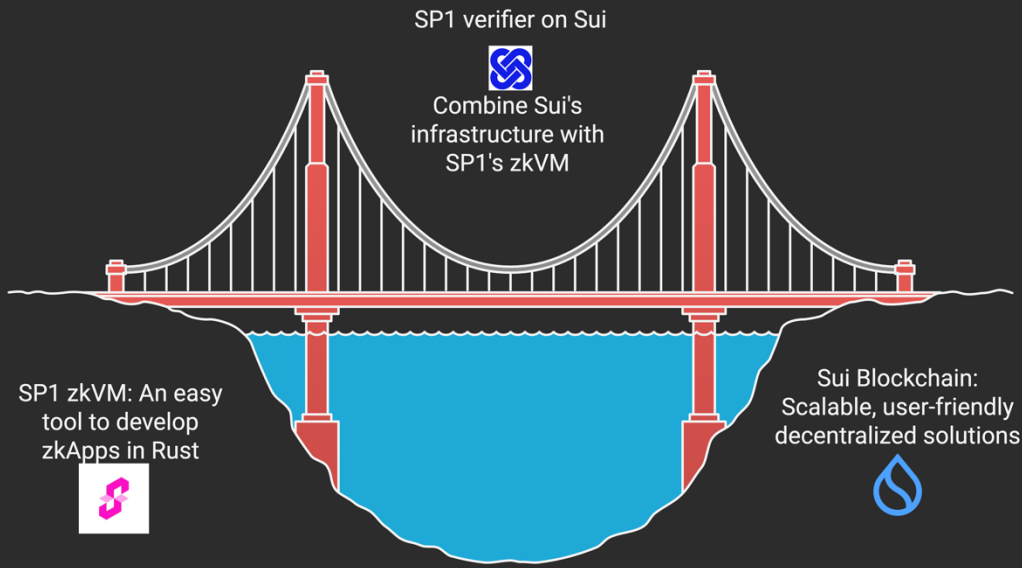
invisible wallets

semi-portable, 2FA



JWT beyond zkLogin

Some complementary ideas



JWT beyond zkLogin

Some complementary ideas

zkLogin

Given a public IDP_pk and zkLogin address:
I have access to a valid JWT under IDP_pk such that:
zkLogin_add = Blake2b256(iss||Poseidon(aud||sub||Poseidon(Salt))) &
Signature on tnx details is valid under epk that is linked to JWT.

New case

Given a public @domain:
I have access to a valid JWT such that:
payload.email = test@domain.xyz

Potential Expansion

Given zkLogin_add and @domain:
I have access to a valid JWT such that:
zkLogin_add = Blake2b256(iss||Poseidon(aud||sub||Poseidon(Salt))) &
payload.email = test@domain.xyz

Thank You!

Some of the slides done by Mysten labs team.